EECS4312 Software Engineering Requirements
Lab 2 (80 points), Version 1
Building ML-based Models to Identify Requirement-Related Code Changes

Instructors: Song Wang
Release Date: Oct 20, 2020

Due: 11:59 PM, Friday, Oct. 30, 2020

In this lab, you are expected to build machine learning based models to identify software 'requirement related code changes' (i.e., code change committed by user/developers for enhancing existing or add new functionalities) among all the commits to help requirement engineers collect requirement update and track the changes between requirement specification and source code. You are expected to apply machine learning techniques, e.g., classification and feature engineering to build more accurate classification models with the dataset provided.

Policies

- You are required to work on your own for this lab.
- When you submit your work, you claim that it is solely the work of you. Therefore, it is considered as an violation of academic integrity if you copy or share any parts of your code during any stages of your development.
- Your (submitted or un-submitted) solution to this assignment (which is not revealed to the public) remains the property of the EECS department. Do not distribute or share your code in any public media (e.g., a non-private Github repository) in any way, shape, or form. The department reserves the right to take necessary actions upon found violations of this policy.
- Emailing your solutions to the instruction or TAs will not be acceptable.

Submitting Your Work

- create a folder named "EECS4312_Lab2".
- put all the result analysis from each part into one PDF file, named **EECS4312_lab2.pdf**.
- create subfolders with names "**part_I**", "**part_II**", and "**part_III**". Put the source code from each part into each folder, e.g., your code from part I should be in "**part_I**".
- make sure the Instructor/TAs can run your code based on the above file structure without any manual modification.

```
zip -r EECS4312_Lab2.zip EECS4312_Lab2
submit 4312 lab2 EECS4312_Lab2.zip
```

Background

Code changes to software occur due to various reasons such as bug fixing (i.e., code changes made for fixing bugs), requirement related (i.e., code changes made for adding new or improving existing functionalities), and code refactoring. Often different changes have different motivations on behalf of the developers. An example code change is shown in Figure 1. Many projects in Github could have hundreds of new code changes everyday, manually classifying these code changes is a time-consuming task. Thus, a machine learning based automatic cod change classification model could be useful for accelerating the development and maintenance of software projects. A typical ML-based classification model is shown in Figure 2.

In this lab, to simplify the task, we only consider identifying 'requirement related code changes', that is in the dataset we only have two class labels for all the changes, i.e., '1' (requirement changes) and '0' (non-requirement changes).

62 solr/core/src/java/org/apache/solr/handler/admin/ConfigSetsHandler.java

```
@@ -170,30 +170,63 @@ private void handleConfigUploadRequest(SolrQueryRequest req, SolrQueryResponse r
170 170
171 171         boolean overwritesExisting = zkClient.exists(configPathInZk, true);
172 172
173     -       if (overwritesExisting && !req.getParams().getBool(ConfigSetParams.OVERWRITE, false)) {
174     -         throw new SolrException(ErrorCode.BAD_REQUEST,
175     -             "The configuration " + configSetName + " already exists in zookeeper");
176     -       }
    173 +       boolean requestIsTrusted = isTrusted(req, coreContainer.getAuthenticationPlugin());
    174 +
    175 +       // Get upload parameters
    176 +       String singleFilePath = req.getParams().get(ConfigSetParams.FILE_PATH, "");
    177 +       boolean allowOverwrite = req.getParams().getBool(ConfigSetParams.OVERWRITE, false);
    178 +       boolean cleanup = req.getParams().getBool(ConfigSetParams.CLEANUP, false);
177 179
178 180         Iterator<ContentStream> contentStreamsIterator = req.getContentStreams().iterator();
```

Figure 1: An example code change.

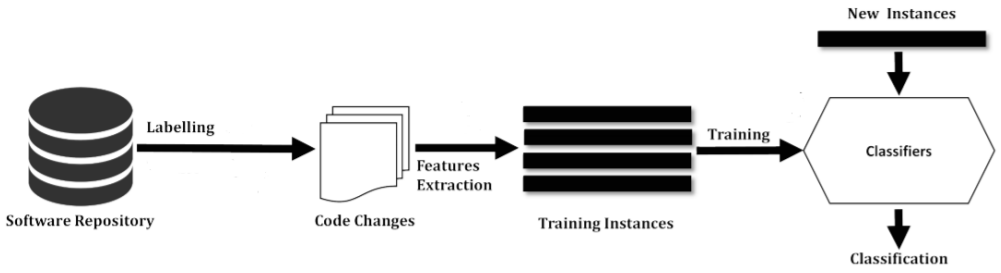

Figure 2: The process of machine learning based classification model for code changes.

The process of a typical code change classification model consists of the following steps:

- **Labelling:** Labelling each change as'1' (requirement changes) and '0' (non-requirement changes) to indicate whether the code change involves requirement update. Labeling change-level defect data requires to further link bug-fixing changes to bug-introducing changes. There are many heuristic approaches to label the ground-truth of each change, in this lab we will provide the label information of each change for you.
- **Feature Extraction:** Extracting the features to represent the changes. Features are attributes extracted from a code change which describe the characteristics of the source code, experience of developer that committed the change, intent of the code change, etc.

- **Model building/training:** Using the features and labels to build and train machine learning based classification models. The widely used classifiers include Naive Bayes, Decision Tree, Random Forest, and Logistic Regression, etc. All these machine learning algorithms are available in Scikit-learn[1] or Weka[2].
- **Classification:** with the well trained classification models we can identify the 'requirement related code changes' among the new changes.

Features/Attributes

This lab provides a basic feature set, i.e., Metadata, to help you start this lab. We will be experimenting with two widely-known open source projects, i.e., Android[3] and Openstack[4].

These data are in **/data** folder.

Metadata: This feature set contains the commit time of a change (i.e., **month_of_year**, **day_of_month**, **day_of_week**, **hour_of_day**, and **minute_of_hour**), the added line count (i.e., **addition**), the deleted line count (i.e., **deletion**), **the commit message of this change**, and revision history on the files involved in this change, i.e., given a change, we collect the number of files in this change that have been revised in the last 30 and 90 days (**changed_file_30** and **changed_file_90**), and the number of revision on all the involved files of this change in the last 30 and 90 days (**total_changed_30** and **total_changed_90**).

Evaluation Metrics

Metrics, i.e., *Precision*, *Recall*, and $F1$, are widely used to measure the performance of classification models. Here is a brief introduction:

$$Precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{1}$$

$$Recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{2}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3}$$

Precision and recall are composed of three numbers in terms of *true positive*, *false positive*, and *false negative*. True positive is the number of predicted requirement-related code changes that are truly requirement changes, while false positive is the number of predicted requirement related ones that are actually not related. False negative records the number of predicted non-requirement changes that are actually requirement related. F1 is the weighted average of precision and recall.

In this lab, you are expected to compare the performance of different classification models by using F1.

(I)- Using Machine Learning Libraries to build a simple model (10 points)

In this lab, you are excepted to build code change classification models by using Python or Java machine learning libraries. All the data we provided use the "csv" data format. The first row of each "csv" data file contains the id of a change, features to represent the change, and the label of the change. The provided features we provided looks like this:

[1] https://scikit-learn.org/stable
[2] https://waikato.github.io/weka-wiki/using_the_api/
[3] https://source.android.com/
[4] https://wiki.openstack.org/

```
Id numeric // *Not a feature* This is the id of a change
owner string
month_of_year numeric
day_of_month numeric
day_of_week      numeric
hour_of_day      numeric
minute_of_hour numeric
deletion numeric
addition numeric
changed_file_30  numeric
changed_file_90 numeric
total_changed_30 numeric
total_changed_90 numeric
isReq {0,1} // *Not a feature* This is label, 1 means requirement related, 0 means non requirment related
```

The data of a change looks like the following:

```
577,1000392,9,21,3,15,27,36,29,0,0,0,0,0
579,1000162,9,21,3,19,57,0,5,0,0,0,0,0
593,1000162,9,21,3,22,2,0,1,0,0,0,0,0
594,1000162,9,21,3,22,2,0,1,0,0,0,0,0
595,1000162,9,21,3,22,2,0,8,0,0,0,0,0
776,1000162,9,21,3,22,2,0,3,0,0,0,0,0
789,1000495,9,22,4,16,29,1,1,0,0,0,0,0
1202,1000392,9,22,4,0,38,0,11,0,0,0,0,0
1205,1001144,9,22,4,3,23,0,4,0,0,0,0,0
1216,1001252,9,22,4,18,43,103,64,0,0,0,0,1
```

Your first task is to build a simple **Naive Bayes** based code change classification models with all the features provided **except the code commit message**. Specifically, for each of the projects, you are required to build and train the **Naive Bayes** based classification models with the provided training and test datasets.

We recommend you use Weka for this lab. Weka is a suite of machine learning software written in Java, which contains a collection of tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces and APIs (Java) for easy access to these functions.

Examples (Java and Python) of using Weka/Scikit-learn libs to build a **Naive Bayes** classifier are available here: **example/**. We also provide example data to drive this example model, which are also in **example/**. Note that, the example classification models use different features and target at a different task.

More useful information about classifiers in the Weka libs:

https://waikato.github.io/weka-wiki/using_the_api/.

In you result report of this part, you are expected to show the performance of **Naive Bayes** based classifiers on different projects by using figures or tables.

For the source code of this part, your code are expected to be executable. The results presented in your report are expected to be replicated by running your code.

(II)- Improving the model by using different ML algorithms (30 points)

Naive Bayes is a simple machine learning classifier, which may deliver poor performance. You are expected to try and find a better machine learning classifier. Note that, each classification algorithm may have several parameters that may affect its performance, you are also expected to tune the parameters for improving the performance of a specific classification algorithm. **You should at least examine three different classifiers**, e.g., Decision Tree (e.g., weka.classifiers.trees.ADTree in Weka), Random Forest (weka.classifiers.trees.RandomForest), and Logistic Regression (weka.classifiers.functions.Logistic), etc.

In your result report of this part, you should show which parameters you have tuned, what are the values you have tried, how the performance of a classifier changes with different parameter values, and the best parameters for each classifier by using figures or tables.

For the source code of this part, your source code are expected to be executable. The results presented in your report are expected to be replicated by running your code.

(III)- Improving bug prediction models with more features (40 points)

The initialized features only contain the basic statistically information of the changes and have not considered the context/commit information of every single change, which might be not enough to build an accurate classification model. You are expected to design new features to improve the best classification model you built in **Task II**.

Some hints for designing new features:

Bag-of-Words/1-Gram: The Bag-of-Words feature set (i.e., 1-gram) is a vector of the count of occurrences of frequent word ($frequency > 3$) in the text. You may want to remove the noisy tokens from the commit messages, which often are not related to intent of a change. After that, you can obtain the bag-of-words features (each token will be a unique feature and you need to combine them with the existing features) from code commit **for both the training and test data**. More details of Bag-of-Words can be found here: https://en.wikipedia.org/wiki/Bag-of-words_model

You are expected to add the above features to the existing feature set for building classification models.

In your result report of this part, you should describe the new features you have collected. Detailed step how you extracted these features (**interacted with your code**). You are expected to rebuild and tune the **best classifier** in Part (II).

For the source code of this part, your code are expected to be executable. The new features and the prediction results presented in your report are expected to be replicated by running your code.